//**

// 2018 ROBOMASTER 技术总结

// 描述: 此文档为全国大学生机器人大赛（机甲大师）北部赛区视觉识别技术

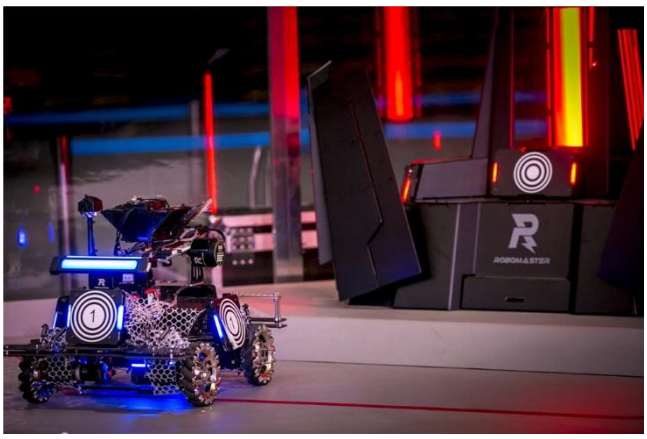
//总结，文档描述了识别步兵装甲的算法。

// 杨道青

// 2018.05.27

//**

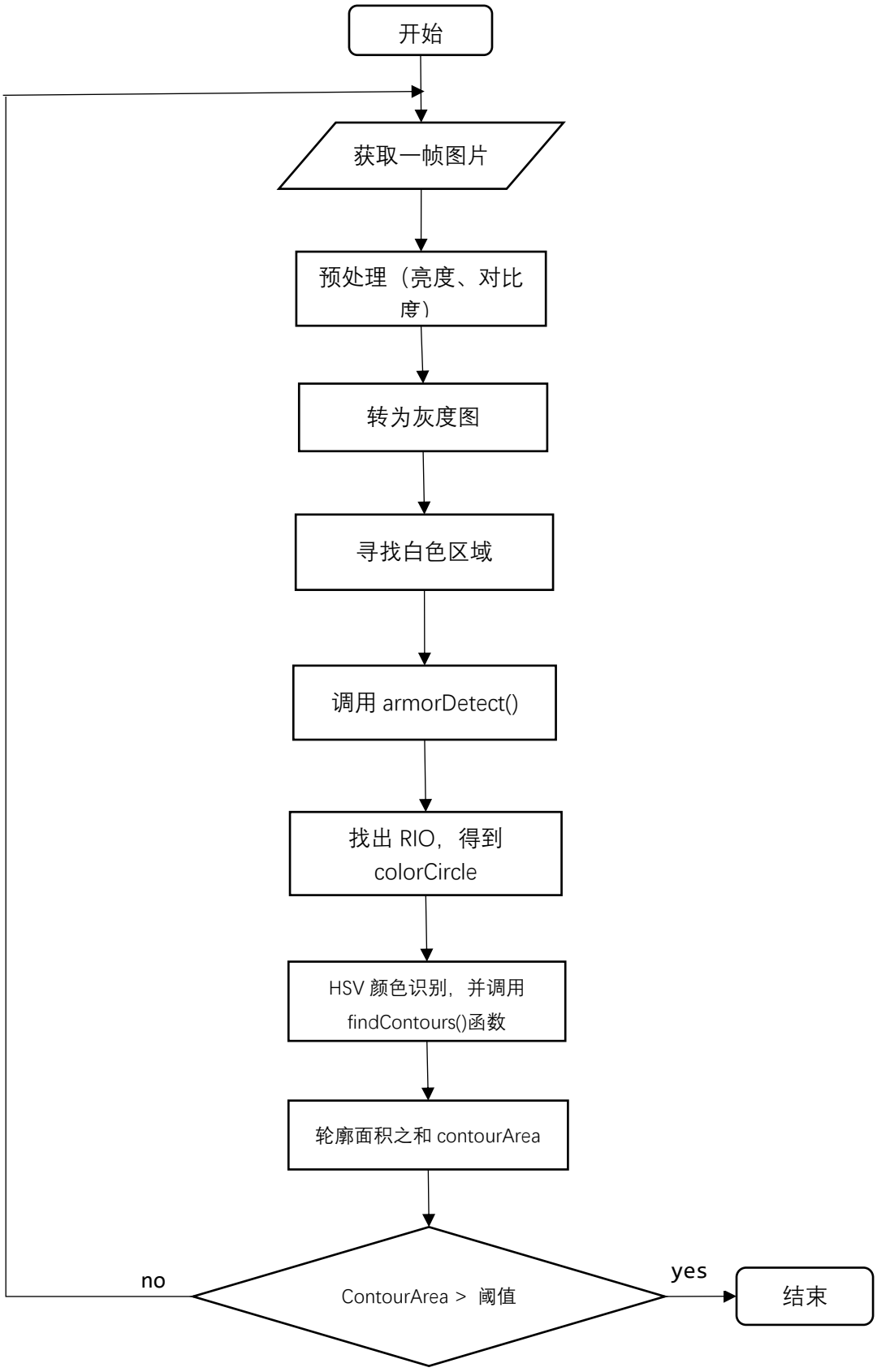
【装甲图片】



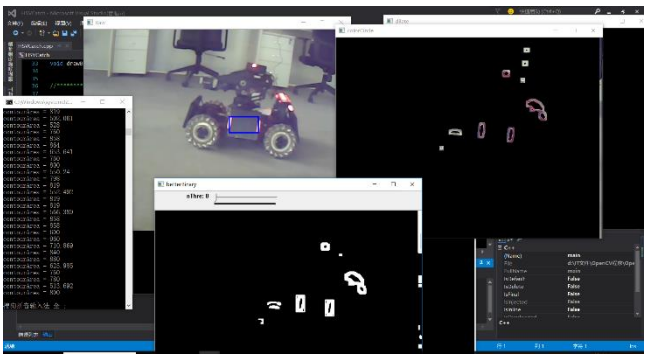
【算法】

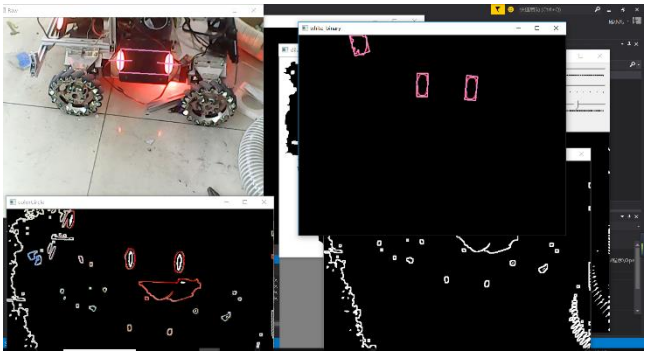
由于装甲灯条亮度过高，摄像头读入的图片显示灯条颜色较浅。故首先对图片预处理（rawImage），即调整对比度、亮度值，凸显白色区域。然后将预处理后的图片转为灰度图，利用像素值寻找白色得到二值图。对二值图寻找轮廓，进行旋转矩形的拟合。之后根据装甲形态特征，封装成 armorDetect( ) 函数标出装甲。设置 ROI，根据装甲位置在原图、二值图、膨胀图分割出装甲，得到 frame0_ArmorROI、binary_ArmorROI、rlt_ArmorROI。接着对 frame0_ArmorROI 做膨胀操作，与 binary_ArmorROI 做减法得到掩模 ErodeMask，再将掩模覆盖到原图，得到装甲周围的颜色区域 colorCircle（为圆环状）。之后把 colorCircle 装换为 HSV 图，利用阈值寻找红（蓝）色，区分敌我。

【流程图】


开始
获取一帧图片
预处理（亮度、对比度）
转为灰度图
寻找白色区域
调用 armorDetect()
找出 RIO，得到 colorCircle
HSV 颜色识别，并调用 findContours()函数
轮廓面积之和 contourArea
ContourArea > 阈值
no
yes
结束

【效果图】





【源码】

```
#include "opencv2/core.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/videoio.hpp"
#include "opencv2/imgproc.hpp"
#include "iostream"
#include "omp.h"
#include "MyComClass.h"
#include "string.h"

using namespace cv;
using namespace std;

//*****************************【全局变量声明】*********************************
#define T_ANGLE_THRE 10
#define T_SIZE_THRE 10

//*****************************【红色】*************************************
int Red_lowH0 = 0;  int Red_highH0 = 10;
int Red_lowH = 156;  int Red_highH = 180;

int Red_lowS = 43;  int Red_highS = 255;
```

```
int Red_lowV = 46; int Red_highV = 255;
//*********************************【蓝色】*************************************
int Blue_lowH = 100;  int  Blue_highH = 124;

int  Blue_lowS = 43;  int  Blue_highS = 255;

int  Blue_lowV = 46; int  Blue_highV = 255;

//*********************************【White阈值】*********************************
int nThre = 83;
int nThreMax = 255;

//RNG rng(12345);
Scalar color = Scalar(255,0,0);

//*****************************************************************************

//*********************************【功能函数声明】*******************************
void brightAdjust(Mat &src, Mat &dst, double dContrast, double dBright); //图像
增强
void getBinaryImage(Mat &src, Mat &dst, int flag, int lowh, int lows, int
lowv,int highh,int highs,int highv); //判断红蓝装甲
void getMask(Mat &src1, Mat &src2, Mat &dst);//制作掩模
void catchWhite(Mat &src, Mat &dst, int nThre);//寻找白色，标出装甲区
vector<RotatedRect> armorDetect(vector<RotatedRect> vEllipse); //检测装甲
void drawBox(RotatedRect box, Mat img); //标记装甲


//*****************************************************************************

int main()
{

    VideoCapture cap0(0);
    //VideoCapture cap0("4.avi");
    //VideoCapture cap0("BlueCar.avi");
    Mat frame0;

    /*PortOperate mPort;
    mPort.Initial("COM4");*/


    Size imgSize;
```

```
    RotatedRect s;   //定义旋转矩形
    vector<RotatedRect> vEllipse; //定以旋转矩形的向量，用于存储发现的目标区域
    vector<RotatedRect> vRlt;

    Mat element = getStructuringElement(MORPH_RECT, Size(3, 3));//5*5的结构元素

    vector<vector<Point> > contour;
    vector<vector<Point> > Armor_Contour;

    cap0 >> frame0;
    imgSize = frame0.size();

    Mat rawImg = Mat(imgSize, CV_8UC3);

    Mat binary = Mat(imgSize, CV_8UC1);
    Mat rlt = Mat(imgSize, CV_8UC1);

    //***************【窗体创建】***********************
    namedWindow("Raw");
    namedWindow("dilate", 1);
    namedWindow("【armor_Binary】", 1);
    namedWindow("二值图", WINDOW_AUTOSIZE);

    //************【创建滑条寻找阈值】******************
    createTrackbar("nThre", "二值图", &nThre, nThreMax);

    while (1)
    {

        if (cap0.read(frame0))
        {
            /*********************
            *cout << "width and height of frame0: " << frame0.rows << "  " <<
frame0.cols << endl;
            *480 x 640
            ******************/

            //*****************************【I-预处理】************************

            brightAdjust(frame0, rawImg, 1.2, -120);//图像亮度调整、对比度1、亮度
-120
            imshow("Bright", rawImg);

            //*****************************【转为灰度图】***********************
```

```
            Mat src,dst;
            rawImg.copyTo(src);
            cvtColor(src, dst, COLOR_BGR2GRAY);
            imshow("srcGrayImage", dst);

            //寻找白色
            catchWhite(dst, binary, nThre);
            imshow("二值图", binary);


            //-------------------------<0>对二值图膨胀-------------------------
            dilate(binary, rlt, element, Point(-1, -1), 3);   //图像膨胀

            //-------------------------<1>寻找轮廓-----------------------------
            findContours(rlt, contour, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE); //
在二值图像中寻找轮廓

            for (int i = 0; i<contour.size(); i++)
            {

                //----------------------【装甲判断】--------------------------
                // 描述：根据轮廓像素点、轮廓面积，初步判断是否符合装甲的灯条
                //
                //------------------------------------------------------------
                if (contour[i].size()> 20 && contour[i].size() < 700)
                {
                        s = fitEllipse(Mat(contour[i]));

                        //画出符合要求的装甲
                        /*drawContours(rlt, contour, i, color, 2, 8);
                        ellipse(rlt, s, Scalar(0, 0, 255), 1, LINE_AA);
                        drawBox(s, rlt);*/
                        imshow("dilate", rlt);

                        vEllipse.push_back(s); //将发现的目标保存

                }

            }

            //调用子程序，在输入的LED所在旋转矩形的vector中找出装甲的位置，并包装成旋
转矩形，存入vector并返回
            vRlt = armorDetect(vEllipse);
```

```
            //----------------------------【准备发送坐标】----------------------
            /*unsigned short tunnelFlag = 1;
            unsigned short x_y;*/

            Point2f vertices[4];      //定义矩形的4个顶点
            Rect armorRect;
            Mat frame_ArmorROI,binary_AromorROI,rlt_ArmorROI,mask_ArmorROI,
colorCircle,rgbBinary;
            Mat bgr[3];

            for (unsigned int nI = 0; nI < vRlt.size(); nI++)
            {
                vRlt[nI].points(vertices);//计算矩形的4个顶点
                armorRect = vRlt[nI].boundingRect();//返回包含旋转矩形的最小矩形

        //**********************【III-颜色判断】*******************************
        //  描述：通过上面寻找到的装甲，设置感兴趣区域，判断装甲颜色，区分敌我状态
        //
        //**************************************************************************
                if (0 <= armorRect.x && 0 <= armorRect.y && (armorRect.x +
armorRect.width) <= frame0.cols && (armorRect.y + armorRect.height) <=
frame0.rows)
                {
                    frame_ArmorROI = frame0(Rect(armorRect.x, armorRect.y,
armorRect.width, armorRect.height));
                    binary_AromorROI = binary(Rect(armorRect.x, armorRect.y,
armorRect.width, armorRect.height));//white-binary
                    rlt_ArmorROI = rlt(Rect(armorRect.x, armorRect.y,
armorRect.width, armorRect.height));//dilated white binary
                    mask_ArmorROI = Mat(frame_ArmorROI.size(), CV_8UC1);
                    rgbBinary = Mat(frame_ArmorROI.size(), CV_8UC1);

                    //----------------------<1>制作标准掩膜mask-----------------
                    getMask(rlt_ArmorROI, binary_AromorROI, mask_ArmorROI);

                    //-----------------------<2>圈出颜色-------------------------
                    frame_ArmorROI.copyTo(colorCircle, mask_ArmorROI);
                    //dilate(colorCircle, colorCircle, element);
                    imshow("colorCircle", colorCircle);//colorCircle为三通道图

                    //---------------------<3>第二次二值化----------------------
                    //   (+)----红色----(+)
```

```
                    //getBinaryImage(colorCircle, rgbBinary,2,Red_lowH,
Red_lowS, Red_lowV,Red_highH, Red_highS, Red_highV);

                    //   (+)----蓝色----(+)
                    getBinaryImage(colorCircle, rgbBinary, 1, Blue_lowH,
Blue_lowS, Blue_lowV, Blue_highH, Blue_highS, Blue_highV);

                    dilate(rgbBinary, rgbBinary, element, Point(-1, -1), 3);
                    findContours(rgbBinary, Armor_Contour, RETR_EXTERNAL,
CHAIN_APPROX_SIMPLE);


                    //*************************【区分敌我】*******************
                    //  描述：将所有轮廓面积求和，大于阈值，即为敌人；否则为队友
                    //
                    //************************************************************
                    double Armor_Contour_Area = 0;
                    if (Armor_Contour.size() >= 2)
                    {
                        for (int i = 0; i < Armor_Contour.size(); i++)
                        {
                            Armor_Contour_Area +=
contourArea(Mat(Armor_Contour[i]));
                        }
                        if (Armor_Contour_Area > 80)// <----阈值
                        {
                            drawBox(vRlt[nI], frame0);//在当前图像中标出装甲的位
置

                           //----------------------坐标发送--------------------
                           //x_y = (unsigned short)((vRlt[nI].center.x) /
 2.5);//横坐标
                           //x_y = ((x_y << 8) | (unsigned
                           short)((vRlt[nI].center.y) / 2.5));//纵坐标
                            //mPort.SendDate(x_y, tunnelFlag);
                        }
                    }


                    imshow("frame_ArmorROI", frame_ArmorROI);
                    imshow("【armor_Binary】", rgbBinary);

                    //刷新
                    //frame_ArmorROI = Scalar::all(0);
```

```
                    binary_AromorROI = Scalar::all(0);
                    rlt_ArmorROI = Scalar::all(0);
                    mask_ArmorROI = Scalar::all(0);
                    rgbBinary = Scalar::all(0);
                }
            }

            imshow("Raw", frame0);

            vEllipse.clear();
            vRlt.clear();

            waitKey(1);
        }
        else
        {
            break;
        }
    }
    cap0.release();
    return 0;

}

void brightAdjust(Mat &src, Mat &dst, double dContrast, double dBright)//图像增强
{
    int rowNumber = dst.rows;
    int colNumber = dst.cols*dst.channels();

    omp_set_num_threads(8);//设置线程的默认周期数为未指定 num_threads 子句的后续并行区域使用
#pragma omp parallel for

    for (int i = 0; i < rowNumber; i++)
    {
        uchar* dstdata = dst.ptr<uchar>(i);
        uchar* srcdata = src.ptr<uchar>(i);
        for (int j = 0; j < colNumber - 1; j++)
        {
            dstdata[j] = saturate_cast<uchar>((dContrast * srcdata[j]) + dBright);
        }
    }
```

```
}



void getBinaryImage(Mat &src, Mat &dst, int flag, int lowh, int lows, int lowv,
int highh, int highs, int highv)
{
    Mat imgHSV;
    vector<Mat> hsvSplit;

    int NumRow = src.rows;
    int NumCol = src.cols;

    cvtColor(src, imgHSV, COLOR_BGR2HSV);

    split(imgHSV, hsvSplit);
    equalizeHist(hsvSplit[2], hsvSplit[2]);
    merge(hsvSplit, imgHSV);


    omp_set_num_threads(8);//设置线程的默认周期数为未指定 num_threads 子句的后续并
行区域使用
#pragma omp parallel for

    for (int r = 0; r < NumRow; r++)
    {
        uchar* tempImage_h = hsvSplit[0].ptr<uchar>(r);
        uchar* tempImage_s = hsvSplit[1].ptr<uchar>(r);
        uchar* tempImage_v = hsvSplit[2].ptr<uchar>(r);
        uchar* temp_imgThre = dst.ptr<uchar>(r);
        for (int c = 0; c < NumCol; c++) {
            if (flag == 1)//判断蓝色
            {
                if ((tempImage_h[c] < highh && tempImage_h[c] > lowh) &&
(tempImage_s[c] < highs && tempImage_s[c] > lows)
                    /*&& (tempImage_v[c] < highv && tempImage_v[c] > lowv)*/)

                {
                    temp_imgThre[c] = 255;
                }
                else
                {
                    temp_imgThre[c] = 0;
                }
```

```
            }
            if (flag == 2)//判断红色（红色颜色区域 h 通道多了一组数）
            {
                if (((tempImage_h[c] < Red_highH0 && tempImage_h[c] >
Red_lowH0) || (tempImage_h[c] < highh && tempImage_h[c] > lowh))
                    && (tempImage_s[c] < highs && tempImage_s[c] > lows)
                    /*&& (tempImage_v[c] < highv && tempImage_v[c] > lowv)*/)

                {
                    temp_imgThre[c] = 255;
                }
                else
                {
                    temp_imgThre[c] = 0;
                }
            }
        }
    }

}

void getMask(Mat &src1, Mat &src2, Mat &dst)
{
    int rowNumber = src1.rows;
    int colNumber = src1.cols;

    omp_set_num_threads(8);
#pragma omp parallel for

    for (int i = 0; i < rowNumber; i++)
    {
        uchar* dstdata = dst.ptr<uchar>(i);
        uchar* src1data = src1.ptr<uchar>(i);
        uchar* src2data = src2.ptr<uchar>(i);
        for (int j = 0; j < colNumber - 1; j++)
        {
            dstdata[j] = (int)(src1data[j] - src2data[j]);
        }
    }
}

void catchWhite(Mat &src, Mat &dst, int nThre)
{
    omp_set_num_threads(8);
```

```
#pragma omp parallel for

        for (int nI = 0; nI<src.rows; nI++)
        {
            uchar* pchar1 = src.ptr<uchar>(nI);
            uchar* pchar2 = dst.ptr<uchar>(nI);
            for (int nJ = 0; nJ <src.cols; nJ++)
            {
                if (pchar1[nJ] > nThre)
                {
                    pchar2[nJ] = 225;
                }
                else
                {
                    pchar2[nJ] = 0;
                }
            }
        }
}

vector<RotatedRect> armorDetect(vector<RotatedRect> vEllipse)
{
    //**********************【存储旋转矩形的四个顶点】**********************
    Point2f ptnI[4],ptnJ[4];
    int i;
    int pt_length1, pt_length2,delta_length;
    for (i = 0; i<4; i++)
    {
        ptnI[i].x = 0; ptnJ[i].x = 0;
        ptnI[i].y = 0; ptnJ[i].y = 0;
    }


    vector<RotatedRect> vRlt;
    RotatedRect armor; //定义装甲区域的旋转矩形
    int nL, nW, delta_y;
    double dAngle;
    //vRlt.clear();
    if (vEllipse.size() < 2) //如果检测到的旋转矩形个数小于2，则直接返回
        return vRlt;
    for (unsigned int nI = 0; nI < vEllipse.size() - 1; nI++) //求任意两个旋转矩
形的夹角
    {
        for (unsigned int nJ = nI + 1; nJ < vEllipse.size(); nJ++)
```

```
        {
            dAngle = abs(vEllipse[nI].angle - vEllipse[nJ].angle);
            while (dAngle > 180)
                dAngle -= 180;//  dAngle in (0,90)

            //判断这两个旋转矩形是否是一个装甲的两个LED等条
            if ((dAngle < T_ANGLE_THRE /*|| 180 - dAngle < T_ANGLE_THRE*/) &&
abs(vEllipse[nI].size.height - vEllipse[nJ].size.height) <  T_SIZE_THRE &&
abs(vEllipse[nI].size.width - vEllipse[nJ].size.width) <  T_SIZE_THRE)
            {
                //------------------(1)取出两个旋转矩形的四个顶点----------------
                vEllipse[nI].points(ptnI); vEllipse[nJ].points(ptnJ);
                //------------------(2)求旋转矩形相邻四个顶点的交叉距离-----------
                pt_length1 = sqrt((ptnI[2].x - ptnJ[0].x) * (ptnI[2].x -
ptnJ[0].x) + (ptnI[2].y - ptnJ[0].y) * (ptnI[2].y - ptnJ[0].y));
                pt_length2 = sqrt((ptnI[3].x - ptnJ[1].x) * (ptnI[3].x -
ptnJ[1].x) + (ptnI[3].y - ptnJ[1].y) * (ptnI[3].y - ptnJ[1].y));
                delta_length = abs(pt_length1 - pt_length2);

                armor.center.x = (vEllipse[nI].center.x +
vEllipse[nJ].center.x) / 2; //装甲中心的x坐标
                armor.center.y = (vEllipse[nI].center.y +
vEllipse[nJ].center.y) / 2; //装甲中心的y坐标
                delta_y = abs(vEllipse[nI].center.y - vEllipse[nJ].center.y);//
纵坐标之差

                armor.angle = (vEllipse[nI].angle + vEllipse[nJ].angle) / 2;
//装甲所在旋转矩形的旋转角度
                if (180 - dAngle < T_ANGLE_THRE)
                    armor.angle += 90;
                nL = (vEllipse[nI].size.height + vEllipse[nJ].size.height) / 2;
//装甲的高度
                nW = sqrt((vEllipse[nI].center.x - vEllipse[nJ].center.x) *
(vEllipse[nI].center.x - vEllipse[nJ].center.x) + (vEllipse[nI].center.y -
vEllipse[nJ].center.y) * (vEllipse[nI].center.y - vEllipse[nJ].center.y)); //装
甲的宽度等于两侧LED所在旋转矩形中心坐标的距离
                //cout << "nW: " << nW << endl;
                if (nW < 150 && nW > 40 && /*delta_y <= (3 * nW /
10)*/(delta_length < 10))//固定装甲两LED的中心距离
                {
                    if (nL < nW)
                    {
                        armor.size.height = nL;
                        armor.size.width = nW;
```

```
                    }
                    else
                    {
                        armor.size.height = nW;
                        armor.size.width = nL;
                    }

                    vRlt.push_back(armor);
                }

            }
        }
    }
    return vRlt;
}

void drawBox(RotatedRect box, Mat img)
{
    Point2f pt[4];
    int i;
    for (i = 0; i<4; i++)
    {
        pt[i].x = 0;
        pt[i].y = 0;
    }
    box.points(pt); //计算二维盒子顶点
    line(img, pt[0], pt[1], color, 2, 8, 0);
    line(img, pt[1], pt[2], color, 2, 8, 0);
    line(img, pt[2], pt[3], color, 2, 8, 0);
    line(img, pt[3], pt[0], color, 2, 8, 0);
}
```